

APPENDIX I:

NLP++ INTEGRATION WITH A KNOWLEDGE BASE

The knowledge base may consist of a hierarchy of concepts (CON). Each concept may have a set of attributes (ATTR). Each attribute may have a name and a value (VAL). The value may be a string (STR), number (NUM), boolean (BOOL), pointer to another concept, or some other type. Each ATTR may have multiple values, and each value may have a distinct type.

Each concept may also have an associated phrase (PHR) of nodes. A node may be similar to a concept in most respects, except, for example, a node may never be placed directly in the knowledge base hierarchy. Rather, nodes may serve as proxies or references to concepts that are in the hierarchy. Phrases may be used to implement idioms, patterns, samples, rules, unordered sets of concepts or any other information.

The objects discussed above, CON, ATTR, VAL and PHR, may be assigned as types of values of NLP++ variables. These functions may treat nodes as concepts.

The functions in **Table XII** enable accessing and manipulating the objects of the knowledge base. These functions illustrate integrating the NLP++ language with a knowledge base to build a text analyzer.

Table XII. Exemplary Functions Associated with Accessing and Manipulating Objects of the Knowledge Base.

	FUNCTION	RETURN TYPE	DESCRIPTION
	FETCH OBJECTS		
5	findroot()	CON	Return the root concept of the knowledge base (named "concept").
	findconcept(<i>parent, name</i>)	CON	Find the named concept under the specified parent concept.
	findconcept(<i>parent, num</i>)	CON	Find the <i>num</i> -th concept under the specified parent concept.
	findattr(<i>con, name</i>)	ATTR	Find the named attribute under the specified concept.
	findattr(<i>con</i>)	ATTR	Fetch the specified concept's list of attributes.
10	attrname(<i>attr</i>)	STR	Fetch the specified attribute's name.
	attrvals(<i>attr</i>)	VAL	Fetch the specified attribute's values.
	findvals(<i>con, name</i>)	VAL	Fetch the values in the specified concept's named attribute.
	numval(<i>con, name</i>)	NUM	Fetch the numeric value of the named attribute.
	strval(<i>con, name</i>)	STR	Fetch the string value of the named attribute.
15	conval(<i>con, attr_str</i>)	CON	Fetch the concept-value of the specified concept's attribute. (The concept-value must be first value).

attrwithval (<i>con</i> , <i>attr_s</i> , <i>val_s</i>)	BOOL	Determine whether the specified attribute has the specified value.
inheritval (<i>con</i> , <i>name</i> , <i>hier</i>)	STR	Find the string value of an attribute having the specified name, searching from the concept <i>con</i> up to the concept <i>hier</i> . (A predetermined special value for <i>hier</i> may specify the root of the knowledge base (KB).)
conceptname (<i>con</i>)	STR	Fetch the name of the specified concept.
conceptpath (<i>con</i>)	STR	Return the entire path of the specified concept as a string.
pathconcept (<i>str</i>)	CON	Fetch the concept specified by the path <i>str</i> .
wordpath (<i>str</i>)	STR	Fetch the entire path of word-concept for the specified string.
findwordpath (<i>str</i>)	STR	Find entire path of word-concept for the specified string. (If not present, don't add the word.)
wordindex (<i>str</i>)	CON	Get index concept that <i>str</i> would be added under.
findhierconcept (<i>name</i> , <i>hier</i>)	CON	Find the named concept in the sub-hierarchy of the specified concept. (A predetermined special value for <i>hier</i> may specify the root of the knowledge base (KB).)

5

10

15

dictfindword (<i>str</i>)	CON	Find the named concept in the dictionary hierarchy of the KB.
attrexists (<i>hier, attr_s, val_s</i>)	BOOL	Determine whether the specified attribute-value pair exists in the specified sub-hierarchy.
attrchange (<i>hier, attr_s, val_s, new_s</i>)	BOOL	Change all the attributes in the specified hierarchy having the specified name and value to have the specified new string value.
down (<i>con</i>)	CON	Fetch the first child of the specified concept.
up (<i>con</i>)	CON	Fetch the parent of the specified concept.
prev (<i>con</i>)	CON	Fetch the left or previous sibling of the specified concept.
next (<i>con</i>)	CON	Fetch the right or next sibling of the specified concept.
nextattr (<i>attr</i>)	ATTR	Fetch the next attribute in a list of attributes.
nextval (<i>val</i>)	VAL	Fetch the next value in a list of values.
getsva (<i>val</i>)	STR	Convert the specified value to a string, if possible.
getstrval (<i>val</i>)	STR	Get string from string-valued <i>val</i> .
getnumval (<i>val</i>)	NUM	Get number from numeric <i>val</i> .
getconval (<i>val</i>)	CON	Get concept from concept-valued <i>val</i> .
MAKE OBJECTS		

5

makeconcept (<i>parent, name, num</i>) makeconcept (<i>parent, name</i>)	CON	Place the named concept under the specified parent concept. If <i>num</i> is non-zero, the named concept becomes the <i>num</i> -th child of the parent. If <i>num</i> is zero or absent, places the named concept at end of the list of children.
addattr (<i>con, attr_s</i>)	ATTR	To the specified concept, add the specified attribute with no value.
addsva (<i>con, name, num</i>)	-	Add the specified numeric value as a string to the specified attribute.
addstrval (<i>con, name, str</i>)	-	Add the specified string value to specified attribute.
addnumval (<i>con, name, num</i>)	-	Add the specified numeric value to the specified attribute.
addconval (<i>con, attr_str, value_con</i>)	-	Add the specified value (<i>value_con</i>) as the value of specified attribute.
getconcept (<i>parent, name</i>)	CON	Find the named concept under the specified parent. If the named concept does not exist, this functions operates as makeconcept (<i>parent, name</i>).
addword (<i>str</i>)	CON	Add the specified word to the dictionary within the KB, if the word is not already present. Also, fetch the dictionary concept for the word.
MOVE & REMOVE		

10

	rmconcept (con)	BOOL	Remove (the entire sub-hierarchy of) the specified concept from the KB.
	rmchild (parent, name)	BOOL	Remove the named child of the specified parent concept.
	rmchild (parent, num)	BOOL	Remove the <i>num</i> -th child of specified parent concept.
	rmvals (con, name)	BOOL	Remove the values of the specified attribute.
5	rmval (attr, val)	BOOL	Remove the specified value from the specified attribute.
	rmattrval (con, attr_s, val_s)	BOOL	Remove the specified string value from the specified attribute.
	rmattr (con, name)	BOOL	Remove the named attribute (and its values) from the specified concept.
	rmchildren (con)	BOOL	Remove the children (and phrase) from the specified concept.
	rmword (word_str)	BOOL	Remove the specified word from the KB dictionary.
10	prunephrases (hier)	BOOL	Remove all phrases from the specified sub-hierarchy.
	replaceval (con, name, str)	-	Replace all of the values of the specified attribute with the specified string.
	replaceval (con, name, num)	-	Replace all of the values of the specified attribute with the specified number.
	replaceval (con, attr_str, value_con)	-	Replace the specified attribute's value with the specified value (<i>value_con</i>).

5

renameconcept (<i>con</i> , <i>name</i>)	-	Rename the specified concept to the specified name.
renamechild (<i>con</i> , <i>num</i> , <i>name</i>)	-	Rename the <i>num</i> -th child of the specified concept to the specified name.
renameattr (<i>con</i> , <i>name</i> , <i>new</i>)	-	Rename the specified attribute to the specified name (<i>new</i>).
moveleft (<i>con</i>)	-	Move the specified concept one to the left in its list.
movecrigh (<i>con</i>)	-	Move the specified concept one to the right in its list.

KNOWLEDGE-BASE PHRASES & NODES

10

findphrase (<i>con</i>)	PHR	Fetch the specified concept's phrase.
sortphrase (<i>con</i>)	-	Alphabetically sort the specified concept's phrase nodes.
phraselength (<i>con</i>)	NUM	Get the number of nodes in the phrase of the specified concept.
nodeconcept (<i>node</i>)	CON	Get the owning concept of the specified node concept.
findnode (<i>phrase</i> , <i>name</i>)	CON	Find the first node with the specified name in the specified phrase.
findnode (<i>phrase</i> , <i>num</i>)	CON	Find the <i>num</i> -th node in the specified phrase.
listnode (<i>node</i>)	CON	Get the first node in the specified node's list.
firstnode (<i>phrase</i>)	CON	Get the first node in the specified phrase.
lastnode (<i>phrase</i>)	CON	Get the last node in the specified phrase.

15

makephrase (<i>con, name</i>)	PHR	Create a phrase in the specified concept by making the named node.
addcnode (<i>con, name</i>)	CON	Make the named node at the end of the specified concept's phrase.
addnode (<i>phrase, name, num</i>)	CON	Make the named node the <i>num</i> -th in the specified phrase.
rmnode (<i>con</i>)	-	Remove the specified node from its phrase.
rmphrase (<i>phrase</i>)	-	Remove the specified phrase from its concept.
rmcphrase (<i>con</i>)	-	Remove the phrase of the specified concept.
renamenode (<i>phrase, name, new</i>)	-	Rename the specified phrase's named node to <i>new</i> .
renamenode (<i>phrase, num, new</i>)	-	Rename the specified phrase's <i>num</i> -th node to <i>new</i> .

APPENDIX II: RULE-FILE ANALYZER

This appendix defines a text analyzer that a shell may use to read
5 pass files of a user-built analyzer. This appendix contains files in the order in
which they are read, the same order in which the rule-file analyzer may be
executed.

The first file, analyzer.seq, defines the sequence of passes in the
analyzer. Each line in that file consists of the name of an algorithm for the
10 pass — for example, "pat," the main pattern-based algorithm. Each line also
contains data associated with the pass. For example, "retok" refers to the
retok pass file associated with the third pass.

The rule-file analyzer uses special functions for constructing the
internal machinery of a text analyzer. Example functions are rfname(),
15 rfaop(), rfastr(), rfarulemark(), rfanonlit(), rfanum(), rfanodes(), rfaarg(), rfaalist(),
rfarange(), rfaexpr(), rfaunary(), rfaopostunary(), rfaargtolist(), rfapair(),
rfalittopair(), rfapairs(), rfaelement(), rfanonlitwit(), rfalitelt(), rfasugg(), rfaelt(),
rfarule(), rfarulelts(), rfarules(), rfaactions(), rfapres(), reaselect(), rfaregion(),
rfaregions(), rfarecurse(), rfarecurses() and rfarulesfile(). They may build an
20 optimized internal semantic representation orthogonal to the semantic
variables that a user may add to parse-tree nodes.

The rule-file analyzer, which analyzes NLP++, is itself defined using
a subset of the full NLP++ language:

25

#####

FILE: ANALYZER.SEQ

#####

#

5 tokenize #
line #
pat retok #
pat bigtok #
pat x_white #
10 pat nlppp #
pat un_mark #
pat list #
pat list1 #
pat gram1 #
15 rec gram2 #
pat preaction #
pat gram4 #
rec gram5 #
pat action #
20 pat pair #
pat pairs #
pat element #
pat rule #
pat rules #
25 pat code #
pat pres #
pat checks #
pat posts #
pat tmp #
30 pat tmp1 #
pat select #
pat region #
pat regions #
pat recurse #
35 pat recurses #
pat rulesfile #
nintern nil #
gen nil #
hash nil #
40 genhash nil #

```
#####
# FILE:      RETOK.PAT
#####
```

```
5 # since RFB rules are hashed, don't need sentinel.
  #@POST
  #      noop()
  #@RULES
  #_XNIL <- _XWILD [fail=(\\)] @@

10  @POST
      rfaname(2)
      single()
  @RULES
15  _CLF [base layer=( _LIT )] <- \\ n [ren=\\n] @@
    _CCR [base layer=( _LIT )] <- \\ r [ren=\\r] @@
    _CHT [base layer=( _LIT )] <- \\ t [ren=\\t] @@

    _CLANGLE [base layer=( _LIT )] <- \\ \\< @@
20  _CPOUND [base layer=( _LIT )] <- \\ \\# @@
    _CDQUOTE [base layer=( _LIT )] <- \\ \\\" @@
    _CATSIGN [base layer=( _LIT )] <- \\ \\@ @@
    _CLPAR [base layer=( _LIT )] <- \\ \\( @@
    _CRPAR [base layer=( _LIT )] <- \\ \\) @@
25  _CCOMMA [base layer=( _LIT )] <- \\ \\, @@
    _CSEMICOLON [base layer=( _LIT )] <- \\ \\; @@
    _CEQUAL [base layer=( _LIT )] <- \\ \\= @@
    _CLBRACKET [base layer=( _LIT )] <- \\ \\[ @@
    _CRBRACKET [base layer=( _LIT )] <- \\ \\] @@
30  _CUNDERSCORE [base layer=( _LIT )] <- \\ \\_ @@
    _CDASH [base layer=( _LIT )] <- \\ \\- @@
    _CSPACE [base layer=( _LIT )] <- \\ \\ \\ @@
    _CRANGLE [base layer=( _LIT )] <- \\ \\> @@

35  _CBEL [base layer=( _LIT )] <- \\ a [ren=\\a] @@
    _CBS [base layer=( _LIT )] <- \\ b [ren=\\b] @@
    _CFF [base layer=( _LIT )] <- \\ f [ren=\\f] @@
    _CVT [base layer=( _LIT )] <- \\ v [ren=\\v] @@
    _CSQUOTE [base layer=( _LIT )] <- \\ \\' @@
40  _CQMARK [base layer=( _LIT )] <- \\ \\? @@
    _CBANG [base layer=( _LIT )] <- \\ \\! @@
    _CDOLLAR [base layer=( _LIT )] <- \\ \\$ @@
    _CPERCENT [base layer=( _LIT )] <- \\ \\% @@
    _CAMPERSAND [base layer=( _LIT )] <- \\ \\& @@
45  _CASTERISK [base layer=( _LIT )] <- \\ \\* @@
    _CPLUS [base layer=( _LIT )] <- \\ \\+ @@
```

```

    _CPERIOD [base layer=( _LIT )] <- \\ \. @@
    _CSLASH [base layer=( _LIT )] <- \\ \/ @@
    _CCOLON [base layer=( _LIT )] <- \\ \: @@
    _CCARET [base layer=( _LIT )] <- \\ \^ @@
5   _CBACKQUOTE [base layer=( _LIT )] <- \\ \` @@
    _CLBRACE [base layer=( _LIT )] <- \\ \{ @@
    _CRBRACE [base layer=( _LIT )] <- \\ \} @@
    _CVBAR [base layer=( _LIT )] <- \\ \| @@
    _CTILDE [base layer=( _LIT )] <- \\ \~ @@
10  _CBSLASH [base layer=( _LIT )] <- \\ \\ @@

```

@POST

excise(1,1)

@RULES

```

15  _xNIL <- \r \n @@

```

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100

```

#####
# FILE:      BIGTOK.PAT
#####

5  @POST
    excise(1, 3)
@RULES
_XNIL <- \# _XWILD \n @@

10 @POST
    excise(1, 2)
@RULES
_XNIL <- \# _XWILD _XEOF @@

15 @POST
    rfastr(2)
    single()
@RULES
_STR [base] <- \" _XWILD \" @@

20 #@POST
#    excise(1, 1)
#@RULES
#_XNIL <- \, [plus] @@

25 # EXPRESSION GRAMMAR.
@POST
    rfaop(1,2)
    single()

30 @RULES
_opAND <- \& \& @@
_opOR  <- \| \| @@
_opINC <- \+ \+ @@
_opDEC <- \- \- @@

35 _opEQ <- \= \= @@
_opNEQ <- \! \= @@
_opGE  <- \> \= @@
_opLE  <- \< \= @@
_opCONF <- \% \% @@

40 _opOUT <- \< \< @@

@RULES
_ENDRULE [base] <- \@ \@ _XWHITE @@

45 #@POST
#    noop()

```

```
#@RULES
#_XNIL <- _xWILD [min=1 max=1 fail=(\@)] @@
```

```
@RULES
```

```
5  _ENDRULE [base] <- \@ \@ _xEOF @@
   _eoPOST [base layer=(\_endMark)] <- \@ \@ POST [t] @@
   _eoCHECK [base layer=(\_endMark)] <- \@ \@ CHECK [t] @@
   _eoPRE [base layer=(\_endMark)] <- \@ \@ PRE [t] @@
   _eoRULES [base layer=(\_endMark)] <- \@ \@ RULES [t] @@
10 _eoRECURSE [base layer=(\_endMark)] <- \@ \@ RECURSE [t] @@
   _eoSELECT [base layer=(\_endMark)] <- \@ \@ SELECT [t] @@
   _eoNODES [base layer=(\_endMark)] <- \@ \@ NODES [t] @@
   _eoMULTI [base layer=(\_endMark)] <- \@ \@ MULTI [t] @@
   _eoPATH [base layer=(\_endMark)] <- \@ \@ PATH [t] @@
15 _eoCODE [base layer=(\_endMark)] <- \@ \@ CODE [t] @@
   _soPOST [base layer=(\_startMark)] <- \@ POST [t] @@
   _soCHECK [base layer=(\_startMark)] <- \@ CHECK [t] @@
   _soPRE [base layer=(\_startMark)] <- \@ PRE [t] @@
   _soNODES [base layer=(\_startMark)] <- \@ NODES [t] @@
20 _soMULTI [base layer=(\_startMark)] <- \@ MULTI [t] @@
   _soPATH [base layer=(\_startMark)] <- \@ PATH [t] @@
   _soCODE [base layer=(\_startMark)] <- \@ CODE [t] @@
   _soSELECT [base layer=(\_startMark)] <- \@ SELECT [t] @@
   _soRECURSE [base layer=(\_startMark)] <- \@ RECURSE [t] @@
25
   # Separating out rule mark so it can be counted.
   # If there are none, then don't need to warn about no rules in pass.
   @POST
       rfarulemark()
30   single()
   @RULES
   _soRULES [base layer=(\_startMark)] <- \@ RULES [t] @@

   @POST
35       rfanonlit(2)
       single()
   @RULES
   _NONLIT [base] <- \_ _xALPHA @@

40 @RULES
   _ARROW [base] <- \< \- @@

   @POST
       rfaname(1)
45   single()
   @RULES
```

```

# Not setting base for these potential keywords.
_LIT <- _xWILD [s one match=(
    N X G P s
    if else while
5    )] @@

_LIT [base] <- _xALPHA @@

@POST
10    rfanum(1)
    single()
@RULES
_NUM [base] <- _xNUM @@

15

```



```

#####
# FILE: NLPPP.PAT
# SUBJ: Creating regions for parsing NLP++ syntax and others.
# NOTE: Code regions are parsed differently from the rules.
5 #####
@POST
    rfanodes(2, "nodes")
    single()
@RULES
10 _NODES [base] <- _soNODES [s] _NONLIT [star] _eoNODES [s opt] @@

@POST
    rfanodes(2, "path")
    single()
15 @RULES
    _PATH [base] <- _soPATH [s] _NONLIT [star] _eoPATH [s opt] @@

@POST
    rfanodes(2, "multi")
20 single()
@RULES
    _MULTI [base] <- _soMULTI [s] _NONLIT [star] _eoMULTI [s opt] @@

@POST
25 group(2,2, "_NLPPP") # An NLP++ region.
    singler(1,3)
@RULES
    _PRES [base unsealed] <-
        _sOPRE [s]
30 _xWILD [fail=(_endMark _startMark)]
        _eoPRE [s opt]
        _xWILD [opt lookahead match=(_endMark _startMark)]
        @@

35 @POST
    group(2,2, "_NLPPP") # An NLP++ region.
    singler(1,3)
@RULES
    _CHECKS [base unsealed] <-
40 _soCHECK [s]
        _xWILD [fail=(_endMark _startMark)]
        _eoCHECK [s opt]
        _xWILD [opt lookahead match=(_endMark _startMark)]
        @@
45 @POST

```

```

    group(2,2, "_NLPPP") # An NLP++ region.
    singler(1,3)
@RULES
    _POSTS [base unsealed] <-
5      _soPOST [s]
      _xWILD [fail=(_endMark _startMark)]
      _eoPOST [s opt]
      _xWILD [opt lookahead match=(_endMark _startMark)]
      @@
10
    @POST
    singler(1,3)
@RULES
    _RULES [base unsealed] <-
15      _soRULES [s]
      _xWILD [fail=(_endMark _startMark)]
      _eorULES [s opt]
      _xWILD [opt lookahead match=(_endMark _startMark)]
      @@
20
    # INI REGION.  FOR RUNNING CODE BEFORE SOMETHING (like @nodes).
    @POST
    group(2,2, "_NLPPP") # An NLP++ region.
    singler(1,3)
25 @RULES
    _INI [base unsealed] <-
      _soINI [s]
      _xWILD [fail=(_endMark _startMark)]
      _eoINI [s opt]
30      _xWILD [opt lookahead match=(_endMark _startMark)]
      @@

    # FIN REGION.  FOR RUNNING CODE AFTER SOMETHING (like @nodes).
    @POST
35      group(2,2, "_NLPPP") # An NLP++ region.
      singler(1,3)
    @RULES
    _FIN [base unsealed] <-
      _soFIN [s]
40      _xWILD [fail=(_endMark _startMark)]
      _eofIN [s opt]
      _xWILD [opt lookahead match=(_endMark _startMark)]
      @@

45 @POST
    group(2,2, "_NLPPP") # An NLP++ region.

```



```
#####
# FILE: UN_MARK.PAT
#####
# Delete empty regions.
5 @NODES _CODE _CHECKS _PRES _POSTS _NODES _PATH _MULTI _RULES

@POST
    excise(1,1)
@RULES
10 _XNIL <- _XWILD [one match=(_startMark _endMark)] @@
```

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1480
 1481
 1482
 1483

#####

FILE: LIST.PAT

#####

@PATH _ROOT _RULES

5

@RECURSE listarg

@POST

rfaarg(1)

10

single()

@RULES

_ARG [base] <- _NONLIT @@

_ARG [base] <- _LIT @@

_ARG [base] <- _STR @@

15

_ARG [base] <- _NUM @@

@@RECURSE listarg

@POST

20

rfalist(2)

single()

@RULES

_LIST [base] <- \(_XWILD [match=(_LIT _NONLIT _STR _NUM) recurse=(listarg)]

\) @@

25

```

#####
# FILE: LIST1.PAT
# SUBJ: For executing in NLP++ regions.
# NOTE: Code-like regions get parsed differently from rule regions.
5 #####
@NODES _NLPPP

@POST
    rfarange(3, 5)
10    singler(2,6)
@RULES
_PREPAIR [base] <-
    \;    # Disambiguating context.
    \< _NUM \, _NUM \> @@
15
_PREPAIR [base] <-
    _xSTART    # Disambiguating context.
    \< _NUM \, _NUM \> @@
20

```

```
#####
# FILE: GRAM1.PAT
# SUBJ: NLP++ sentence and expression grammar.
# NOTE: RECURSIVE PASS.
```

```
5 #####
```

```
@NODES _NLPPP
```

```
@RULES
```

```
10 # NLP++ KEYWORDS.
```

```
_IF [base] <- if [s] @@
```

```
_ELSE [base] <- else [s] @@
```

```
_WHILE [base] <- while [s] @@
```

```
15 # Binary ops.
```

```
@POST
```

```
    movesem(1)
```

```
    single()
```

```
@RULES
```

```
20 _OP <- _xWILD [s one match=( _opAND _opOR
```

```
    _opEQ _opNEQ _opGE _opLE
```

```
    _opCONF _opOUT    )]
```

```
    @@
```

```
25
```

```

#####
# FILE: GRAM2.PAT
# SUBJ: NLP++ code syntax.
# NOTE: RECURSIVE PASS.
5 #####
@NODES _NLPPP

# Catch the start of a function call here, so it won't be grabbed by
# expression grammar.
10 @POST
    fncallstart()
    single()
@RULES
_VARLIST [base] <-
15     _XWILD [s one match=( S G N X P ) layer=( _VARNAME )]
        \ ( @@

@POST
    fncallstart()
20     single()
@RULES
_FNCALLLIST [base] <- _LIT [layer=( _FNNAME )] \ ( @@

@POST
25     movesem(2)          # Move expr semantic object up the tree.
        single()
@RULES

_EXPR <- \ ( _XWILD [s one match=( _EXPR _NUM _STR )] \ ) @@
30
@POST
    rfaexpr(1,2,3)
    singler(1,3)
@RULES
35 _EXPR <-
    _XWILD [s one match=( _EXPR _NUM _STR )]
    _XWILD [s t one match=( \* \ / \% _opCONF )]
    _XWILD [s one match=( _EXPR _NUM _STR )]
    _XWILD [s one fail=( _opINC _opDEC )]
40     @@

# Handling precedence. That's why these rules look funny.
@POST
    rfaexpr(1,2,3)
45     singler(1,3)
@RULES

```



```

_EXPR <-
  _XWILD [s one match=( _EXPR _NUM _STR )]
  _XWILD [s t one match=( \+ \- )]
  _XWILD [s one match=( _EXPR _NUM _STR )]
5  _XWILD [s one match=( _XANY _XEND _XEOF ) except=( \/ \* \%
      _opCONF _opINC _opDEC )]
  @@

@POST
10  rfaexpr(1,2,3)
    singler(1,3)

@RULES
_EXPR <-
  _XWILD [s one match=( _EXPR _NUM _STR )]
15  _XWILD [s t one match=( \< \> _opLE _opGE _opEQ _opNEQ )]
  _XWILD [s one match=( _EXPR _NUM _STR )]
  _XWILD [s one match=( _XANY _XEND _XEOF ) except=( \/ \* \% \+ \-
      _opCONF _opINC _opDEC )]
  @@

20  @POST
    rfaexpr(1,2,3)
    singler(1,3)

@RULES
25  _EXPR <-
    _XWILD [s one match=( _EXPR _NUM _STR )]
    _XWILD [s t one match=( _opAND _opOR )]
    _XWILD [s one match=( _EXPR _NUM _STR )]
    _XWILD [s one match=( _XANY _XEND _XEOF )
30      except=( \/ \* \% \+ \- \< \> _opLE _opGE _opEQ _opNEQ
          _opCONF _opINC _opDEC )]
    @@

# Making assignment into an expr.
35 # LOWEST PRECEDENCE of any operator except output op (<<).

_EXPR <-
  _VAR [s]
  \= [s]
40  _XWILD [s one match=( _EXPR _NUM _STR )]
  _XWILD [s one match=( _XANY _XEND _XEOF )
      except=( \/ \* \% \+ \- \< \> _opLE _opGE _opEQ _opNEQ
          _opAND _opOR
          _opCONF
45      \=          # To associate right to left.
          _opINC _opDEC )]

```

@@

Output operator.

LOWEST PRECEDENCE of any operator.

5

_EXPR <-

_XWILD [s one match=(_STR _EXPR)]

_opOUT [s]

_XWILD [s one match=(_EXPR _NUM _STR)]

10

_XWILD [s one match=(_XANY _XEND _XEOF)

except=(\ / * \% \+ \- \< \> _opLE _opGE _opEQ _opNEQ

_opAND _opOR

_opCONF

\=

15

_opINC _opDEC)]

@@

@POST

rfaunary(1,2)

20

singler(1,2)

@RULES

Unary operators.

Highest precedence, apart from post operators.

25

_EXPR <- _XWILD [s one match=(_opINC _opDEC)]

_VAR [s]

_XWILD [s one match=(_XANY _XEND _XEOF) except=(_opINC _opDEC)]

@@

30

_EXPR <- \! [s]

_XWILD [s one match=(_EXPR _NUM _STR)]

_XWILD [s one match=(_XANY _XEND _XEOF) except=(_opINC _opDEC)]

@@

35

Highest precedence operators.

@POST

rfapostunary(1,2)

single()

@RULES

40

_EXPR <-

_VAR [s]

_XWILD [s one match=(_opINC _opDEC)]

@@

45

Post unary ops have precedence.

@POST

```
rfaunary(2,3)
singler(2,3)
```

```
@RULES
```

```
# Only do this if you're at the start of something or there's an
# operator to the left.
```

```
_EXPR <-
```

```
_XWILD [s one match=( _xSTART \< \> \+ \- \* \/ \% \! \=
      _opINC _opDEC _opLE _opGE _opEQ _opNE _opAND _opOR
      _opCONF
      _opOUT
      )]
```

```
_XWILD [s t one match=( \- \+ )]
```

```
_XWILD [s one match=( _EXPR _NUM )]
```

```
_XWILD [s one match=( _XANY _XEND _XEOF )]
```

```
except=( _opINC _opDEC)]
```

```
@@
```

```

# GENERALIZED FUNCTION CALL GRAMMAR.
# -----
# LIST GRAMMAR.
# FUNCTION CALL GRAMMAR.
5  @POST
    addarg(1,3)
    listadd(1,3)
@RULES

10  _VARLIST <- _VARLIST
    \, [opt]
    _XWILD [one match=( _EXPR _NUM _STR)]
    _XWILD [one match=( \, \) )]      # lookahead.
    @@

15  @POST
    addarg(1,3)
    listadd(1,3)
@RULES

20  _XNIL <- _FNCALLLIST
    \, [opt]
    _XWILD [one match=( _EXPR _NUM _STR)]
    _XWILD [one match=( \, \) )]      # lookahead.
    @@

25  @POST
    varfn()
    single()
@RULES

30  _VAR [layer=( _EXPR)] <- _VARLIST \) @@

    @POST
        movesem(1)
        single()

35  @RULES
    _FNCALL [layer=( _EXPR)] <- _FNCALLLIST \) @@

```

```
#####
# FILE: PREACTION.PAT
#####
@NODES _NLPPP
5
@POST
    preaction()
    single()
@RULES
10 _ACTION [base] <- _PREPAIR _FNCALL [s] \; [s opt] @@
```

0
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1480
 1481
 1482
 1483
 1484
 1

```
#####
# FILE: GRAM4.PAT
# SUBJ: NLP++ syntax.
# NOTE: RECURSIVE PASS.
5 #####
@NODES _NLPPP

@POST
    movesem(2)
10    single()
@RULES

_IFPART <- _IF _XWILD [s one match=( _EXPR _NUM _STR )] @@

15 # Simple statements.
@POST
#    movesem(1)
    makestmt(1)
    single()
20 @RULES
_STMT <- _XWILD [s one match=( _EXPR _NUM _STR )] \; [s] @@

# EMPTY STATEMENT.
@RULES
25 _STMT <- \; [s] @@
```

```
#####
# FILE: GRAM5.PAT
# SUBJ: NLP++ syntax.
# NOTE: RECURSIVE PASS.
```

```
5 #####
```

```
@NODES _NLPPP
```

```
@POST
```

```
10     makestmt(1)
       single()
```

```
@RULES
```

```
# NEED THE BASE, OR GRAMMAR INFINITE LOOP!
```

```
15 _STMTS [base] <- _XWILD [s one match=( _STMT _EXPR
    _BLOCK
    )] @@
```

```
@POST
```

```
20     addstmt(1, 2)
       single()
```

```
@RULES
```

```
    _STMTS [base] <- _STMTS _XWILD [s one match=( _STMT _EXPR
    _BLOCK
25     )] @@
```

```
@POST
```

```
       movesem(2)
       single()
```

```
30 @RULES
```

```
_BLOCK <- \{ [s] _STMTS \} [s] @@
```

```
# EMPTY BLOCK.
```

```
@RULES
```

```
35 _BLOCK <- \{ [s] \} [s] @@
```

```
@POST
```

```
       ifstmt(1, 2)
       single()
```

```
40 @RULES
```

```
_IFSTMT <-
    _IFPART
    _XWILD [s one match=( _BLOCK _STMT _EXPR)]
45     @@
```

```
# WHILE STATEMENT
```

```

@POST
    movesem(2)
    single()
@RULES
5  _WHILECOND <- _WHILE _EXPR @@

    # Should make sure expr is parenthesized.
@POST
    whilestmt(1, 2)
10  single()
@RULES

    _STMT <- _WHILECOND    _XWILD [s one match=( _BLOCK _STMT)]
    @@

15  @POST
    #    movesem(2)
    makestmt(2)
    single()
20  @RULES
    _ELSEPART <-
        _ELSE
        _XWILD [s one match=( _BLOCK _STMT _EXPR)]
        @@

25  @POST
    ifelse(1, 2)
    single()
@RULES
30  _STMT <-
    _IFSTMT _ELSEPART
    @@

@POST
35  movesem(1)
    singler(1, 1)
@RULES
    _STMT [base] <- _IFSTMT
    _XWILD [s one match=( _XANY _XEND _XEOF ) except=( _ELSE )]
40  @@

```



```

#####
# FILE:      PAIR.PAT
#####
@PATH _ROOT _RULES
5
@RECURSE listarg

@POST
    rfaarg(1)
10    single()
@RULES
    _ARG [base] <- _NONLIT @@
    _ARG [base] <- _LIT @@
    _ARG [base] <- _STR @@
15    _ARG [base] <- _NUM @@

@@RECURSE listarg

@RECURSE argtolist
20
@POST
    rfaargtolist(1)
    single()
@RULES
25    _LIST <- _ARG @@

@@RECURSE argtolist

@POST
30    rfapair(1, 3)
    single()
@RULES
    _PAIR [base] <- _LIT \= [trig] _XWILD [min=1 max=1 match=( _LIT _NONLIT
        _STR _NUM _LIST) recurse=(listarg argtolist)] @@
35

```

```
#####
# FILE:      PAIRS.PAT
#####
```

```
@PATH _ROOT _RULES
```

5

```
@RECURSE littopair
```

```
@POST
```

```
    rfalittopair(1)
```

10

```
    single()
```

```
@RULES
```

```
_PAIR <- _LIT @@
```

```
@@RECURSE littopair
```

15

```
@POST
```

```
    rfapairs(2)
```

```
    single()
```

```
@RULES
```

20

```
_PAIRS [base] <- \[ _xWILD [match=( _LIT _PAIR \*) recurse=(littopair)] \]
```

```
@@
```

```
#####
# FILE:      ELEMENT.PAT
#####
@PATH _ROOT _RULES
5
@POST
    rfaelement(1, 2)
    single()
@RULES
10 _ELEMENT [base] <- _NONLIT _PAIRS @@
    _ELEMENT [base] <- _LIT _PAIRS @@
    _ELEMENT [base] <- _NUM _PAIRS @@
```

#####

```

#####
# FILE:      RULE.PAT
#####
@PATH _ROOT _RULES
5
@RECURSE rulelt

@POST
    rfanonlitelt(1)
10    single()
@RULES
    _ELEMENT [base] <- _NONLIT @@

@POST
15    rfalitelit(1)
    single()
@RULES
    _ELEMENT [base] <- _LIT @@
    _ELEMENT [base] <- _NUM @@
20
@@RECURSE rulelt

@RECURSE sugg

@POST
25    rfasugg(1)
    single()
@RULES
    _SUGG <- _ELEMENT @@
30
@@RECURSE sugg

@RECURSE elt

@POST
35    rfaelt(1)
    single()
@RULES
    _ELT <- _ELEMENT @@
40
@@RECURSE elt

@RECURSE rulelts

@POST
45    rfarulelts(1)

```

```

        single()
@RULES
    _PHRASE [base] <- _ELT [plus] @@

5  @@RECURSE ruleelts

    @POST
        rfarule(1, 3)
        single()
10 @RULES
    _RULE [base] <-
        _XWILD [one match=( _NONLIT _ELEMENT _LIT) recurse=(ruleelt sugg)]
        _ARROW [trig]
        _XWILD [recurse=(ruleelt elt ruleelts) fail=( _ENDRULE _ARROW)]
15 _ENDRULE
    @@

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```
#####
# FILE:      RULES.PAT
#####
@PATH _ROOT _RULES
5
@POST
    rfarules(1)
    single()
@RULES
10 _RULES [base] <- _RULE [plus trig] @@
```

```
#####
# FILE:      RULES.PAT
#####
@PATH _ROOT _RULES
5
@POST
    rfarules(1)
    single()
@RULES
10 _RULES [base] <- _RULE [plus trig] @@
```



```
#####
# FILE: PRES.PAT
#####
5 @PATH _ROOT _PRES _NLPPP
  @POST
    rfapres(1)
    single()
  @RULES
10 _PRES [base] <- _ACTION [plus] @@
```

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100

```
#####
# FILE: CHECKS.PAT
#####
@PATH _ROOT _CHECKS _NLPPP
5
@POST
    rfaactions(1)
    single()
@RULES
10 _CHECKS [base] <- _STMTS [plus] @@
```

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100

```
#####
# FILE: POSTS.PAT
#####
@PATH _ROOT _POSTS _NLPPP
5
@POST
    rfaactions(1)
    single()
@RULES
10 _POSTS [base] <- _STMTS [plus] @@
```

```
#####
# FILE: TMP.PAT
# SUBJ: Delete the holding rules nodes.
#####
5 @NODES _ROOT

@POST
    splice(1,1)
@RULES
10
    _XNIL <- _RULES @@
    _XNIL <- _CODE @@
    _XNIL <- _PRES @@
    _XNIL <- _CHECKS @@
15 _XNIL <- _POSTS @@
```

```
#####
# FILE: TMP.PAT
# SUBJ: Delete the holding rules nodes.
#####
5 @NODES _ROOT

@POST
    splice(1,1)
@RULES
10
    _XNIL <- _RULES @@
    _XNIL <- _CODE @@
    _XNIL <- _PRES @@
    _XNIL <- _CHECKS @@
15 _XNIL <- _POSTS @@
```

```
#####
# FILE: TMP1.PAT
# SUBJ: splice the NLPPP container.
#####
5 @NODES _ROOT

@POST
    splice(1,1)
@RULES
10
_XNIL <- _NLPPP @@
```

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100

```
#####
# FILE:      SELECT.PAT
#####
@POST
5      rfaselect(2)
      single()
@RULES
_SELECT [base] <- _soSELECT [opt] _NODES _eoSELECT [opt] @@
_SELECT [base] <- _soSELECT [opt] _MULTI _eoSELECT [opt] @@
10 _SELECT [base] <- _soSELECT [opt] _PATH _eoSELECT [opt] @@
```

```
#####
# FILE:      REGION.PAT
#####
@POST
5      rfaregion(1, 2, 3, 4)
      single()
@RULES
_REGION [base] <- _PRES [opt] _CHECKS [opt] _POSTS [opt] _RULES @@

10
```

```
#####
# FILE:      REGIONS.PAT
#####
@POST
5      rfaregions(1)
      single()
@RULES
      _REGIONS [base] <- _REGION [plus] @@

10
```

#####


```
#####
# FILE:      RECURSE.PAT
#####
@POST
5      rfarecurse(2, 3, 5)
      single()
@RULES
_RECURSE [base] <- _sorecurse [s] _LIT _REGIONS [opt] _eorecurse [s] _LIT
[opt] @@
10
```

```
#####
# FILE:      RECURSES.PAT
#####
@POST
5      rfarecurses(1)
      single()
@RULES
_RECURSES [base] <- _RECURSE [plus] @@
```

10

#####

```
#####
# FILE:      RULESFILE.PAT
#####
@POST
5      rfarulesfile(1, 2, 3, 4)
      single()
@RULES
# ALLOWING EMPTY RULE REGION IF THERE IS A CODE REGION.
_RULESFILE [base] <- _CODE _SELECT [opt] _RECURSES [opt] _REGIONS [opt] @@
10 _RULESFILE [base] <- _CODE [opt] _SELECT [opt] _RECURSES [opt] _REGIONS @@
```

APPENDIX III: A BNF GRAMMAR FOR AN INSTANTIATION OF NLP++

This appendix specifies a syntax for an embodiment of NLP++. The syntax is given in extended Backus-Naur form:

```

<_RULESFILE> ::= [ <_CODE> ] [ <_SELECT> ] [ <_RECUSES> ] [ <_REGIONS> ]
<_RECUSES>  ::= { <_RECURSE> }
<_RECURSE>  ::= <_SORECURSE> <_LIT> [ <_REGIONS> ] <_eoRECURSE> [ <_LIT> ]
10 <_REGIONS> ::= { <_REGION> }
<_REGION>   ::= [ <_PRES> ] [ <_CHECKS> ] [ <_POSTS> ] [ <_RULES> ]
<_SELECT>   ::= [ <_SOSELECT> ] { <_NODES> | <_MULTI> | <_PATH> } [ <_eoSELECT> ]
<_CODE>     ::= <_soCODE> { <_STMTS> } [ <_eoCODE> ]
<_PRES>     ::= <_soPRE> { <_ACTION> } [ <_eoPRE> ]
15 <_CHECKS>  ::= <_soCHECK> { <_STMTS> } [ <_eoCHECK> ]
<_POSTS>    ::= <_soPOST> { <_STMTS> } [ <_eoPOST> ]
<_RULES>    ::= <_soRULES> { <_RULE> } [ <_eoRULES> ]

<_RULE>     ::= <_SUGG> <_ARROW> <_PHRASE> <_ENDRULE>
20 <_SUGG>    ::= <_ELT>
<_PHRASE>   ::= { <_ELT> }
<_ELT>      ::= <_ELEMENT> | <_TOKEN>
<_ELEMENT>  ::= <_TOKEN> <_PAIRS>
<_TOKEN>    ::= <_NONLIT> | <_LIT> | <_NUM>
25 <_PAIRS>   ::= "[" { <_LIT> | <_PAIR> } "]"
<_PAIR>     ::= <_LIT> "=" <_VAL>
<_VAL>      ::= <_LIT> | <_NONLIT> | <_STR> | <_NUM> | <_LIST>
<_LIST>     ::= "(" { <_ARG> } ")"
<_ARG>      ::= <_LIT> | <_NONLIT> | <_STR> | <_NUM>
30
<_STMTS>    ::= { <_STMT> | <_EXPR> | <_BLOCK> }
<_BLOCK>    ::= "{" <_STMTS> "}"

<_STMT>     ::= <_IFSTMT> [ <_ELSEPART> ]
35 <_IFSTMT>  ::= <_IFPART> ( <_BLOCK> | <_STMT> | <_EXPR> )
<_IFPART>   ::= <_IF> ( <_EXPR> | <_NUM> | <_FLOAT> | <_STR> )
<_ELSEPART> ::= <_ELSE> ( <_BLOCK> | <_STMT> | <_EXPR> )

```

```

<_STMT>      ::= <_WHILECOND> ( <_BLOCK> | <_STMT> | <_EXPR> ) ";"
<_WHILECOND> ::= <_WHILE> <_EXPR>

```

```

<_STMT>      ::= ( <_EXPR> | <_NUM> | <_FLOAT> | <_STR> )

```

5

```

<_ACTION>    ::= <_PREPAIR> <_FNCALL> ";"
<_EXPR>      ::= <_FNCALL> | <_VAR>
<_FNCALL>    ::= [ <_SCOPE> ] <_LIT> "(" [ <_FNARGLIST> ] ")"
<_SCOPE>     ::= <_LIT> ":" ":"
10 <_FNARGLIST> ::= <_FNARG> { "," <_FNARG> }
<_FNARG>     ::= <_EXPR> | <_NUM> | <_FLOAT> | <_STR>
<_VAR>       ::= ( "S" | "G" | "N" | "X" ) "(" [ <_FNARGLIST> ] ")"

```

```

15 <_EXPR>    ::= "(" <_FNARG> ")"
<_EXPR>      ::= <_FNARG>
              ( "*" | "/" | "%" | <_opCONF> | "+" | "-" | "<" | ">"
                | <_opLE> | <_opGE> | <_opEQ> | <_opNEQ> | <_opAND> | <_opOR> )
              <_FNARG>

```

```

20 <_EXPR>    ::= <_VAR> "=" <_FNARG>
<_EXPR>      ::= ( <_STR> | <_EXPR> ) <_opOUT> <_FNARG>
<_EXPR>      ::= ( <_opINC> | <_opDEC> ) <_VAR>
<_EXPR>      ::= "!" <_FNARG>
<_EXPR>      ::= <_VAR> ( <_opINC> | <_opDEC> )

```

```

25 <_EXPR>    ::= ( "-" | "+" ) ( <_EXPR> | <_NUM> | <_FLOAT> )

```

```

<_IF>        ::= "if"
<_ELSE>      ::= "else"
<_WHILE>     ::= "while"

```

30

```

<_PREPAIR>   ::= "<" <_NUM> "," <_NUM> ">"

```

```

<_NODES>     ::= <_soNODES> { <_NONLIT> } [ <_eoNODES> ]
<_PATH>      ::= <_soPATH> { <_NONLIT> } [ <_eoPATH> ]

```

```

35 <_MULTI>   ::= <_soMULTI> { <_NONLIT> } [ <_eoMULTI> ]

```

```

<_opAND>     ::= "&" "&"
<_opOR>      ::= "|" "|"

```

<_opINC> ::= "+" "+"
 <_opDEC> ::= "-" "-"
 <_opEQ> ::= "=" "="
 <_opNEQ> ::= "!=" "
 5 <_opGE> ::= ">" ">"
 <_opLE> ::= "<" "<"
 <_opCONF> ::= "%" "%"
 <_opOUT> ::= "<" "<"

10 <_ENDRULE> ::= "@" "@"

<_eoPOST> ::= "@" "@ "POST"
 <_eoCHECK> ::= "@" "@ "CHECK"
 <_eoPRE> ::= "@" "@ "PRE"
 15 <_eoRULES> ::= "@" "@ "RULES"
 <_eoRECURSE> ::= "@" "@ "RECURSE"
 <_eoSELECT> ::= "@" "@ "SELECT"
 <_eoNODES> ::= "@" "@ "NODES"
 <_eoMULTI> ::= "@" "@ "MULTI"
 20 <_eoPATH> ::= "@" "@ "PATH"
 <_eoCODE> ::= "@" "@ "CODE"

<_soPOST> ::= "@" "POST"
 <_soCHECK> ::= "@" "CHECK"
 25 <_soPRE> ::= "@" "PRE"
 <_soRULES> ::= "@" "RULES"
 <_soRECURSE> ::= "@" "RECURSE"
 <_soSELECT> ::= "@" "SELECT"
 <_soNODES> ::= "@" "NODES"
 30 <_soMULTI> ::= "@" "MULTI"
 <_soPATH> ::= "@" "PATH"
 <_soCODE> ::= "@" "CODE"

<_NONLIT> ::= "_" <_XALPHA>
 35 <_LIT> ::= <_XALPHA>
 <_LIT> ::= "\\\" <_XPUNCT>
 <_FLOAT> ::= <_XNUM> "." [<_XNUM>
 <_NUM> ::= <_XNUM>

<_ARROW> ::= "<" "-"

<_STR> is a string token.

<_XALPHA> is an alphabetic token.

5 <_XNUM> is an integer token.

<_XPUNCT> is a punctuation character token.

APPENDIX IV: THE CONFIDENCE OPERATOR ACCORDING TO ONE EMBODIMENT

The confidence operator combines confidence values while never exceeding 100% confidence. The confidence operator provides a way to accumulate evidence for competing hypotheses.

Mathematical functions are available or may be with the properties that infinite evidence (or some maximal quantity) equates to 100% confidence and that no evidence equates to a 0% confidence. In one embodiment:

10

$$P = 100 * (1 - 1/(1 + E)) \quad (1)$$

$$E = P (100 - P) \quad (2)$$

where P is the percentage of confidence and E is a fabricated evidence metric that ranges from 0 to infinity.

Say, the suffix "-ence" gives a 70% confidence level that a word is a noun. Say, also, that if a word appears immediately following the word "a," there is an 85% confidence that the word is a noun. Then the accumulated confidence is 70 % 85, some number greater than 85 and less than 100.

20

The probability of a noun, based on the suffix, is 70%. Equation (2) gives E_1 , the evidence from the suffix, as 2.33. The probability of a noun, based on the preceding article "a," is 85%. Equation (2) gives E_2 , the evidence from the article, as 5.66. E, the total evidence, is the sum of E_1 and E_2 . Thus, E is 8.00. Equation (1) gives a probability of 88.9% with evidence E equal to 8.00.

25

E_1 may be based on statistical studies, a guess (educated or otherwise), gut feel, etc. The same is true of E_2 . This is a standing problem in statistics. While there are many ways to generate the initial confidence numbers, typically, one starts with initial values and modifies them based on how well those values work in practice.

United States Patent & Trademark Office
Office of Initial Patent Examination — Scanning Division



Application deficiencies found during scanning:

☐ Page(s) _____ of _____ were not present
for scanning. (Document title)

☐ Page(s) _____ of _____ were not present
for scanning. (Document title)

Pages 40 - 95 of the specification are appendix.

☒ Scanned copy is best available. Drawing figures 3-12 are dark